Acceleration Data Structures

Michael Doggett
Department of Computer Science
Lund university

LUGG
Lund University Graphics Group

Ray tracing

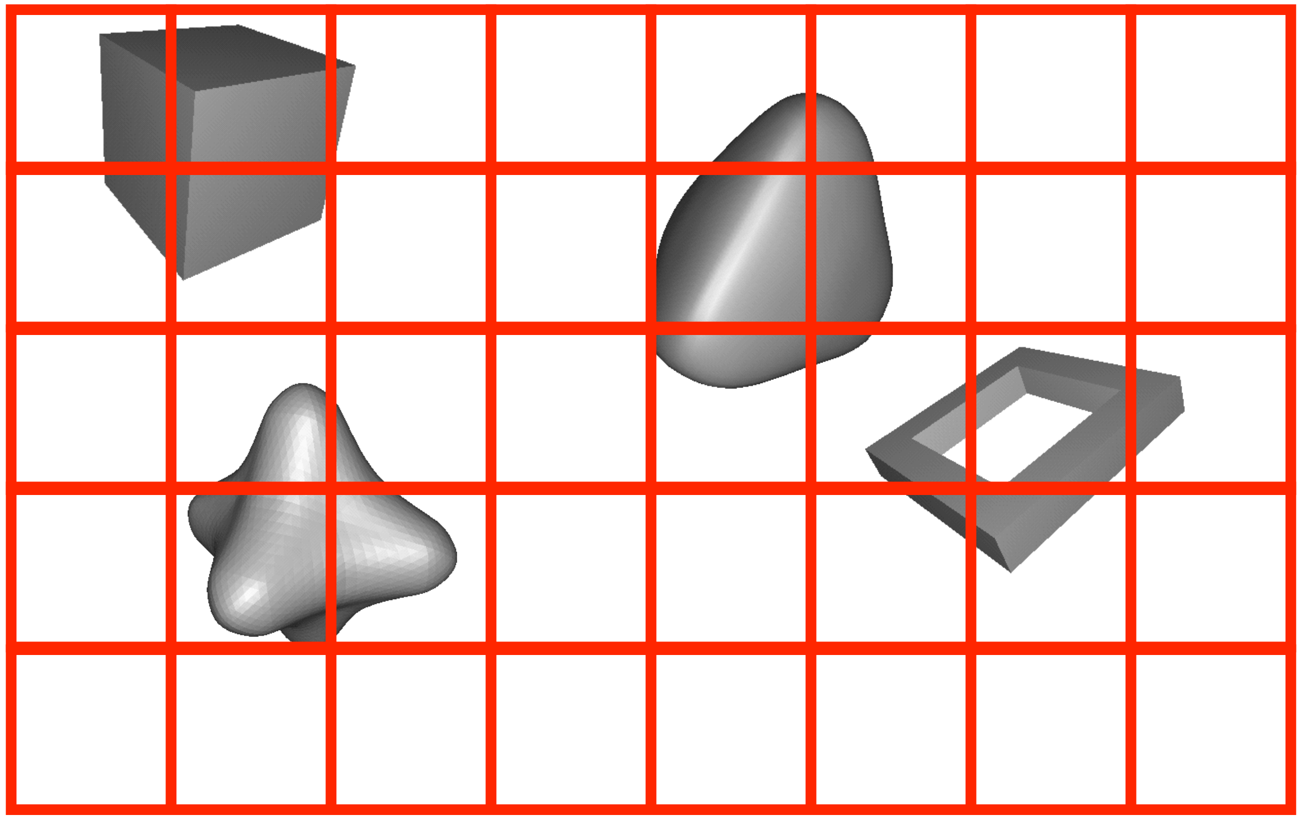
- So far
  - ray tracing
  - sampling
  - object intersections
- Today
  - How do we make it faster?
  - Performance = rays x objects
  - Text book, chapter 9 BVH

Spatial data structures

- What is it?
  - Data structure that organizes geometry in 2D or 3D or higher
  - The goal is faster processing
  - Needed for most ”speed-up techniques”
    - Faster real-time rendering
    - Faster intersection testing
    - Faster collision detection
    - Faster ray tracing and global illumination
- Games use them extensively
- Movie production rendering tools always use them too

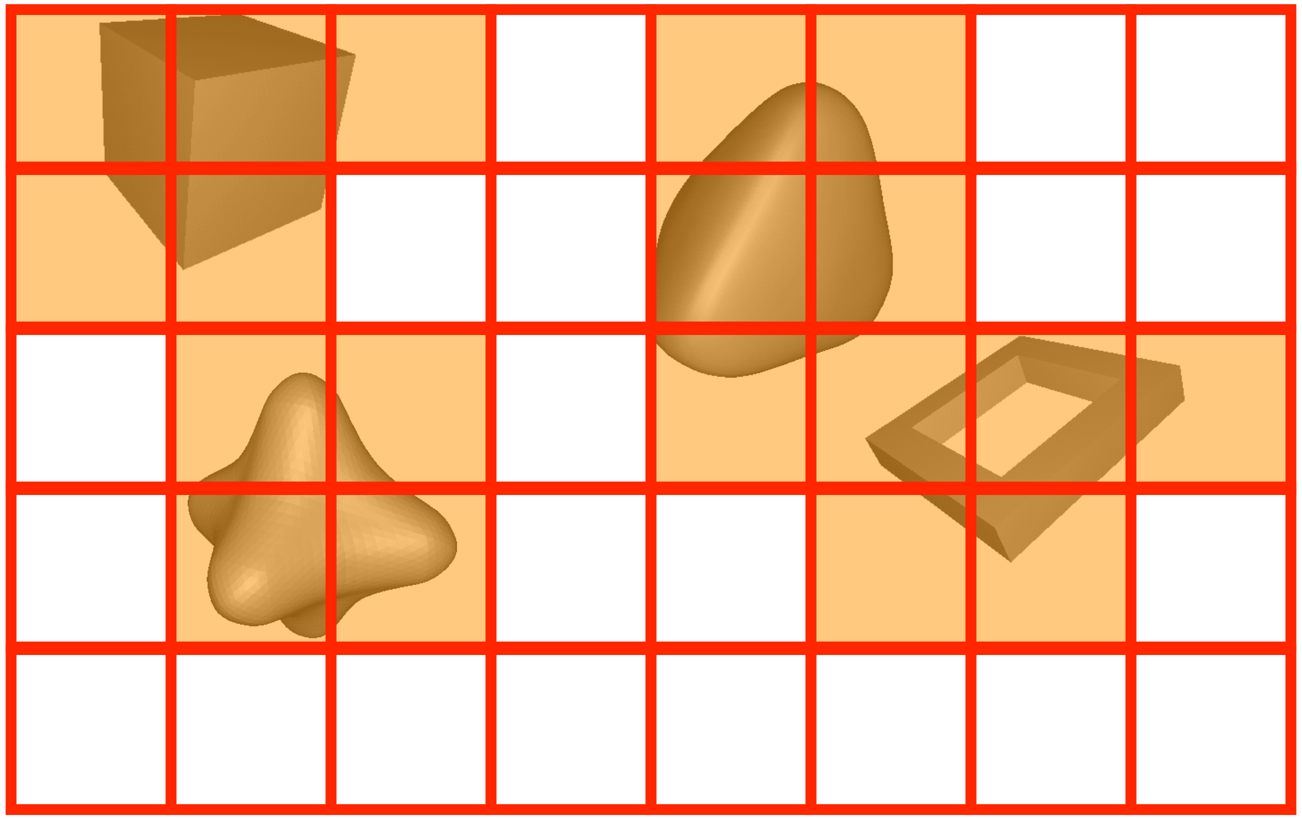
Uniform Grids

- Positives
  - Easy to build
  - Easy to update
- Negatives
  - Could use a lot of memory
  - What grid size?



Building Uniform Grids

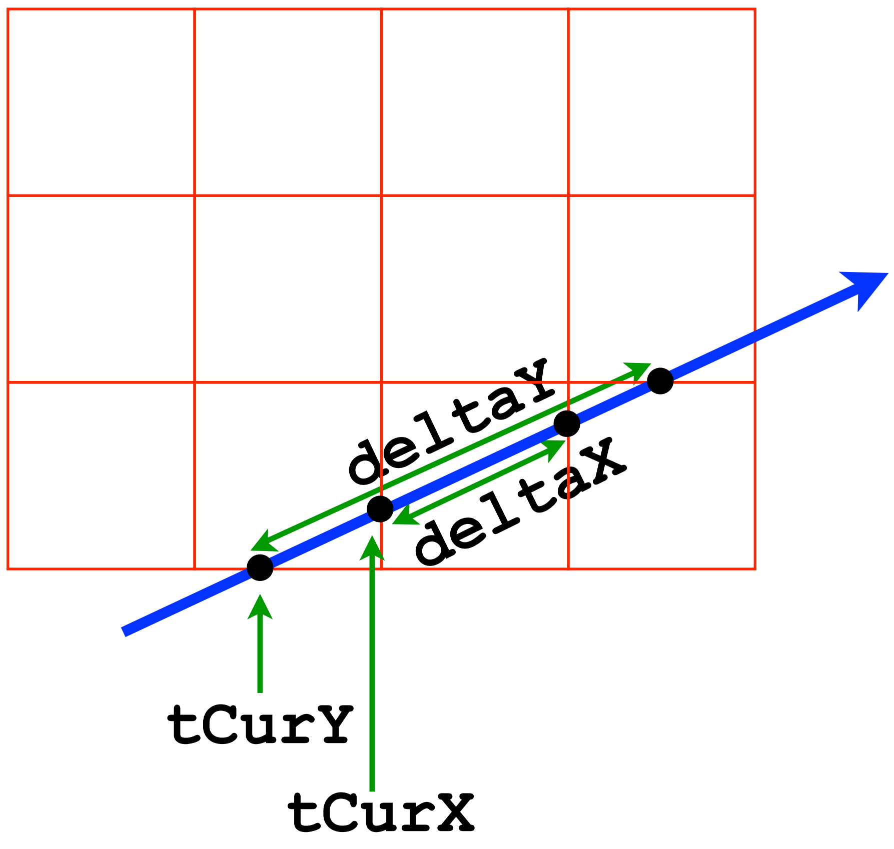
1. Create bounding box
2. break up into equal sized units
3. For each unit the object overlaps, insert a pointer



Uniform Grid Traversal

- Use a 3D DDA algorithm
- E.g. Amanatides' Fast voxel traversal

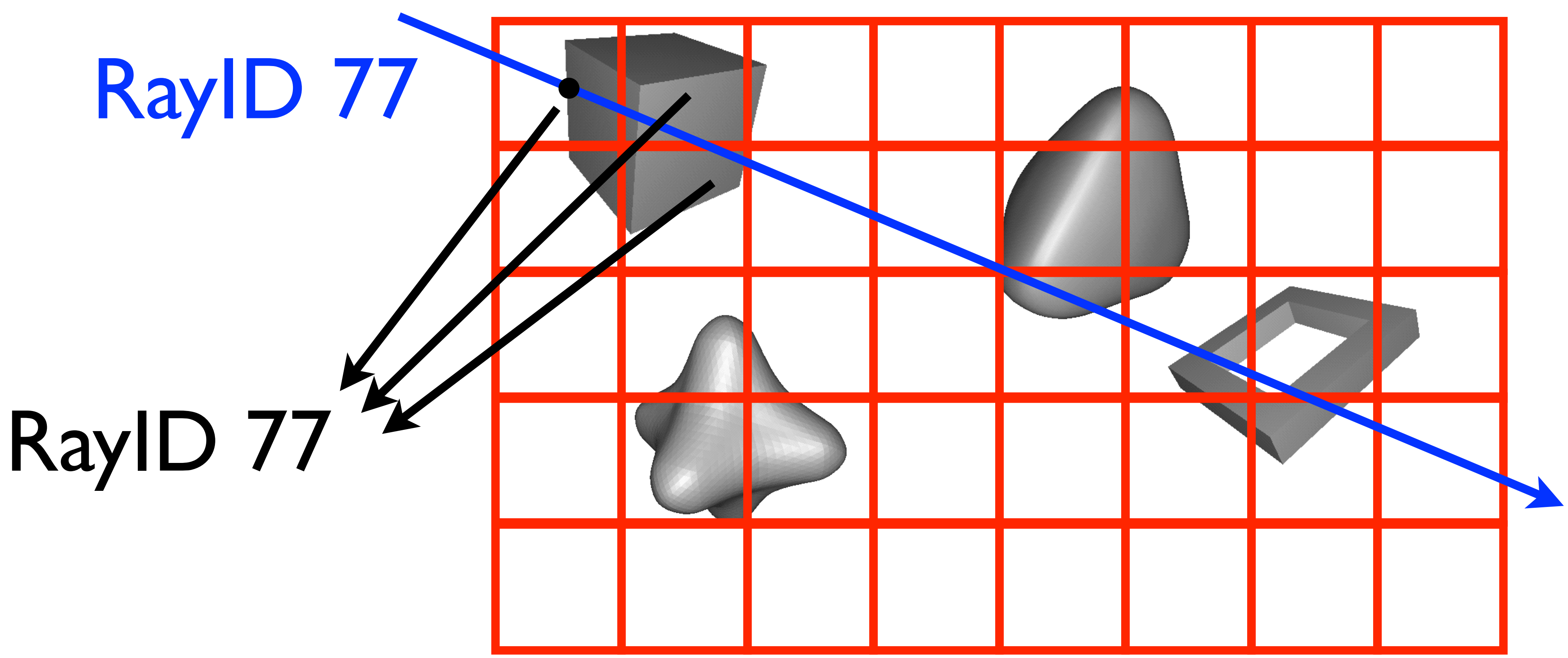
```
if(tCurX<tCurY) {
  tCurX += deltaX; X += 1;
} else {
  tCurY += deltaY; Y += 1;
} NextVoxel(X, Y);
```



N.B. this is simple positive case, could be stepping in negative direction

Uniform Grid Traversal Problem

- Same ray can intersect same object multiple times
- Worst case - ground polygon
- Solution : Object stores most recent RayID

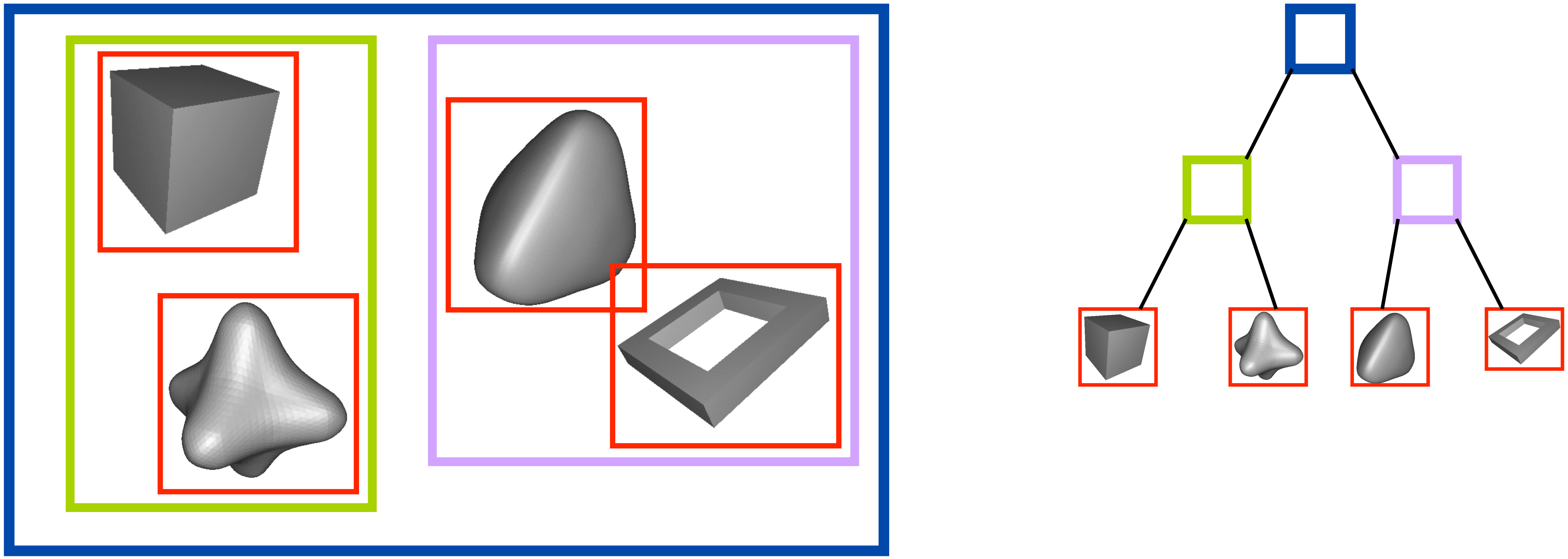


Are there more adaptive ways?

- Organize geometry into a hierarchy

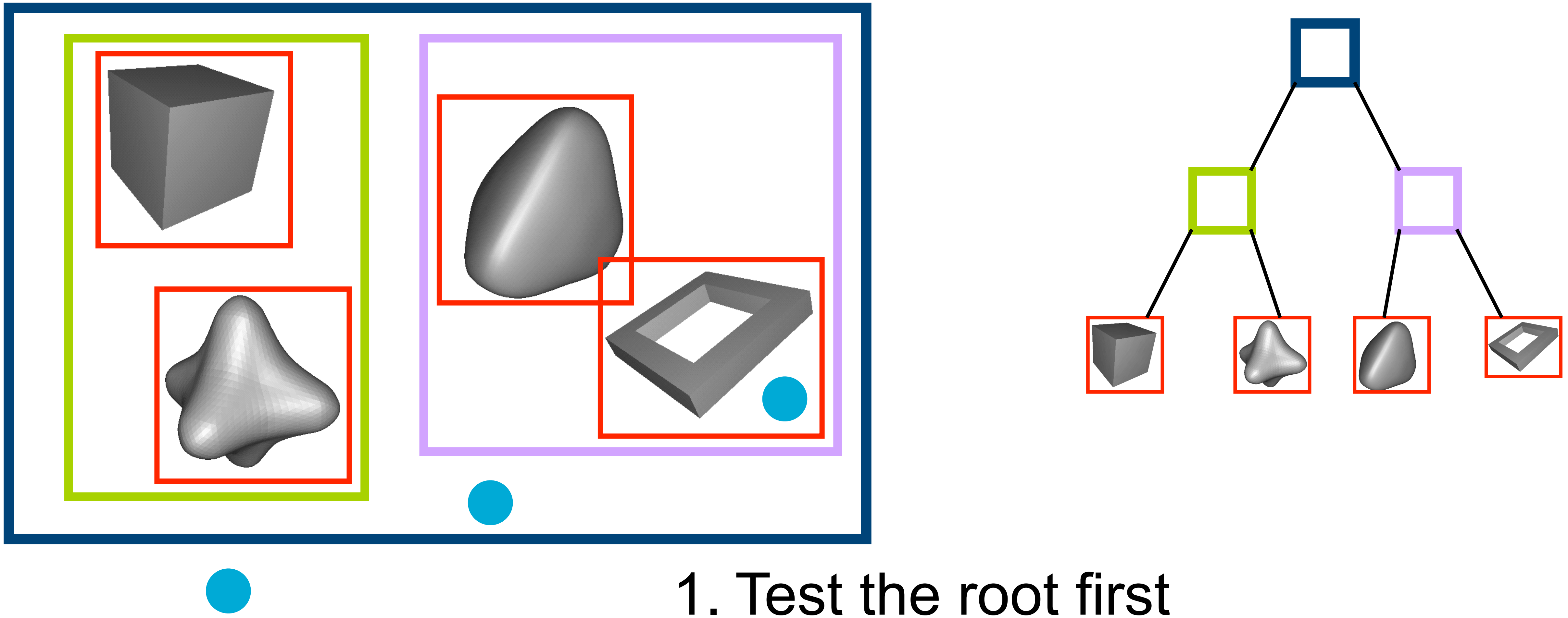
In 2D space

Data structure



What's the point? An example

- Assume we click on screen, and want to find which object we clicked on



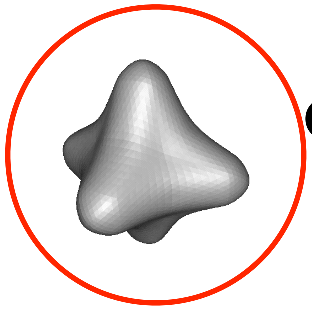
click!

1. Test the root first
2. Descend recursively as needed
3. Terminate traversal when possible

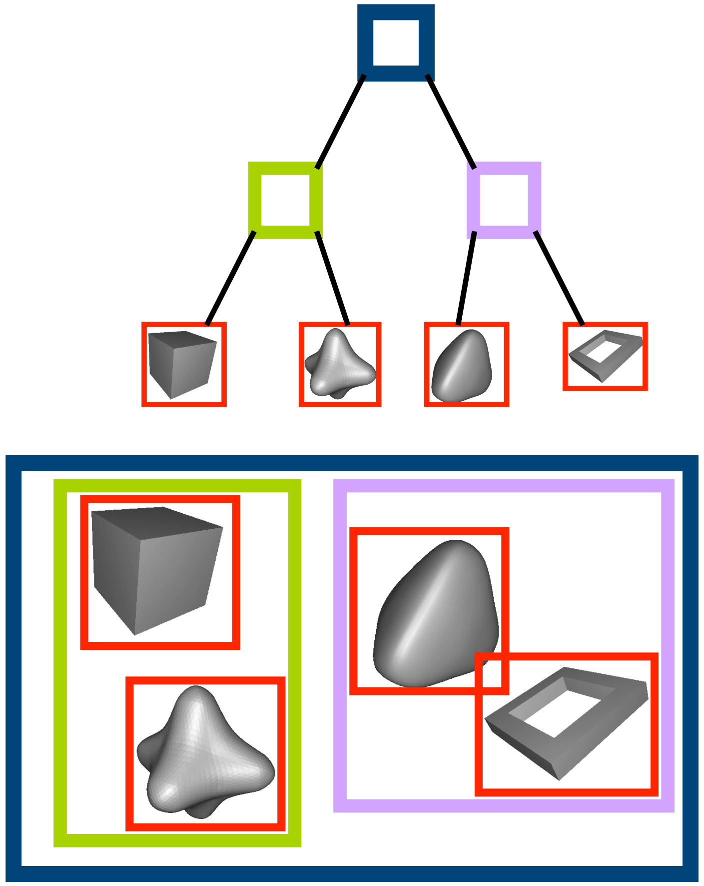
In general: get O(log n) instead of O(n)

Bounding Volume Hierarchy (BVH)

- Most common bounding volumes (BVs):
  - Sphere
  - Boxes (AABB and OBB)



- The BV does not contribute to the rendered image -- rather, encloses an object
- The data structure is a *k*-ary tree
  - Leaves hold geometry
  - Internal nodes have at most *k* children
  - Internal nodes hold BVs that enclose all geometry in its subtree

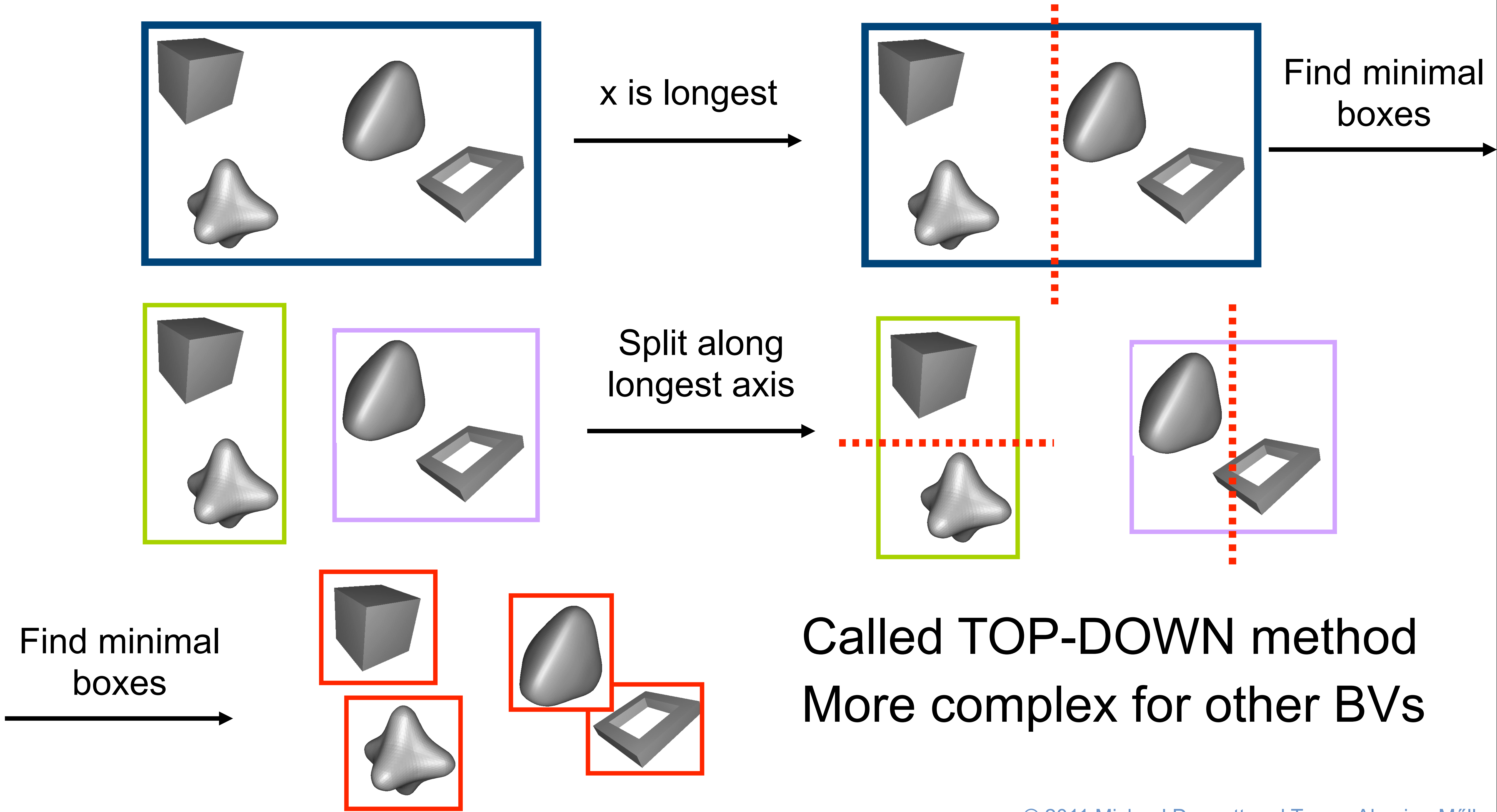


Some facts about trees

- *Height of tree, h,* is longest path from root to leaf
- *A balanced tree* has all leaves at height h or $h+1$
- Height of balanced tree with n nodes: $\text{floor}(\log_k(n))$
- Binary tree (k=2) is the simplest
  - k=4 and k=8 is quite common for computer graphics as well

How to create a BVH? Example: BV=AABB

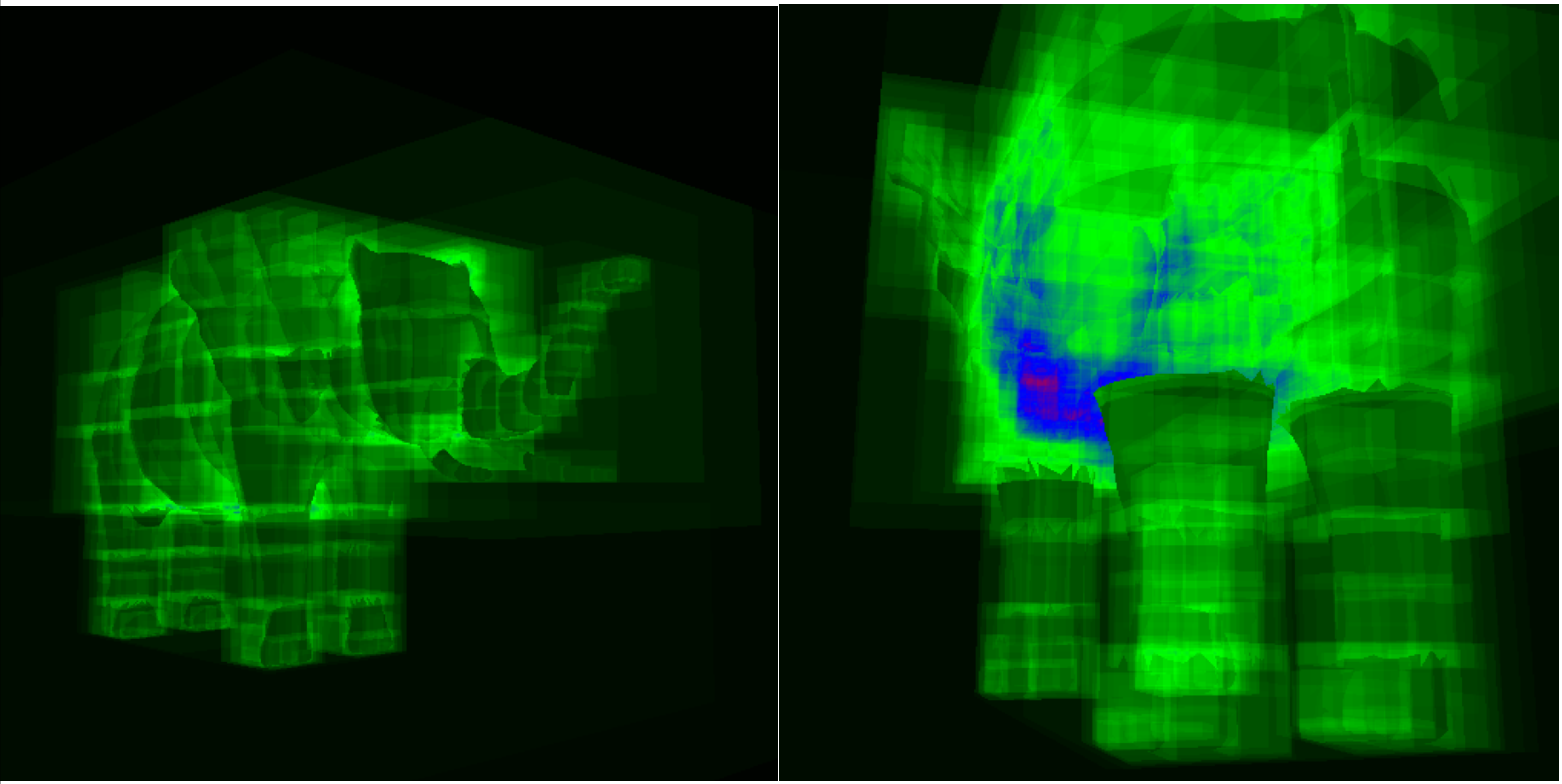
- Find minimal box, then split along longest axis



Called TOP-DOWN method

More complex for other BVs

BVH node visits



Stopping criteria for Top-Down creation

- Need to stop recursion some time…
  - Either when BV is empty
  - Or when only one primitive (e.g. triangle) is inside BV
  - Or when $<n$ primitives is inside BV
  - Or when recursion level l has been reached
  - Even better if it's cost based. (e.g. stop when splitting doesn't improve cost)

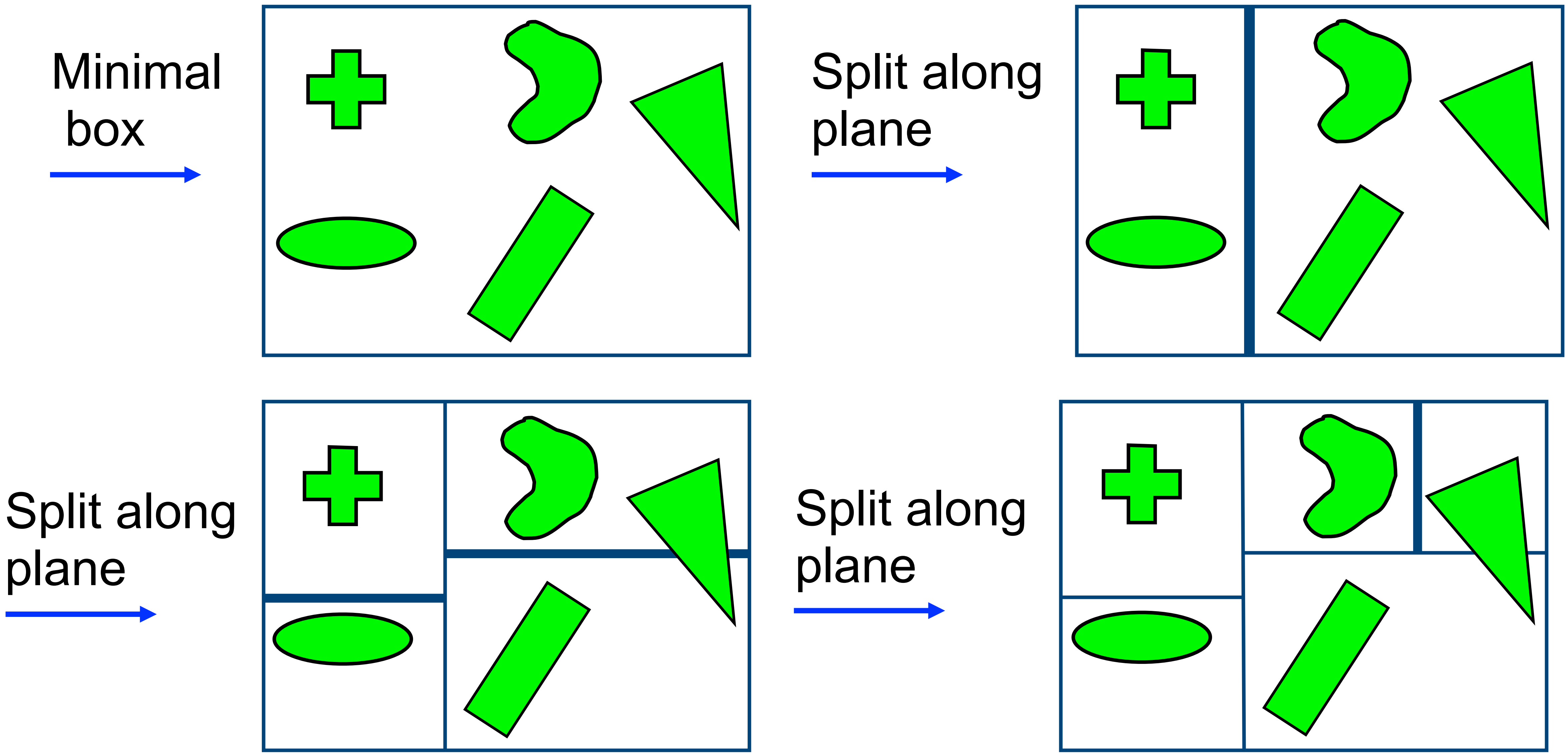
- Similar criteria for BSP trees and octrees

Binary Space Partitioning (BSP) Trees

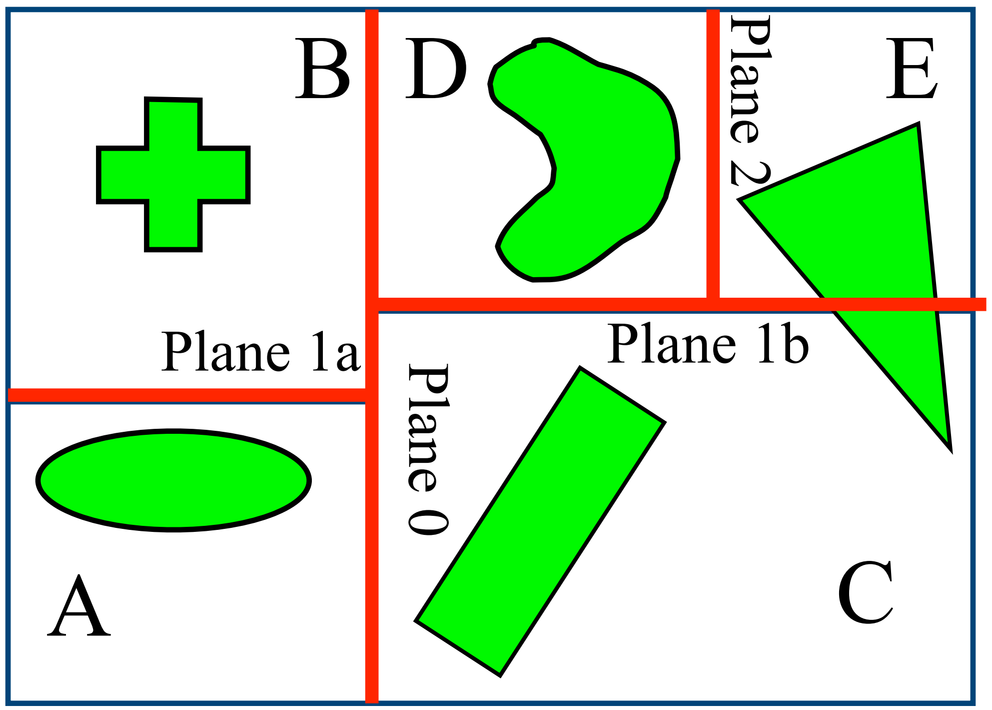
- Two different types:
  - Axis-aligned - kd-tree
    - kd-tree usually alternates between axes when splitting, $x,y,z,x,\ldots$
  - Polygon-aligned
- General idea:
  - Divide space with a plane
  - Sort geometry into the space it belongs
  - Done recursively
- If traversed in a certain way, we can get the geometry sorted along an axis
  - Exact for polygon-aligned
  - Approximately for axis-aligned

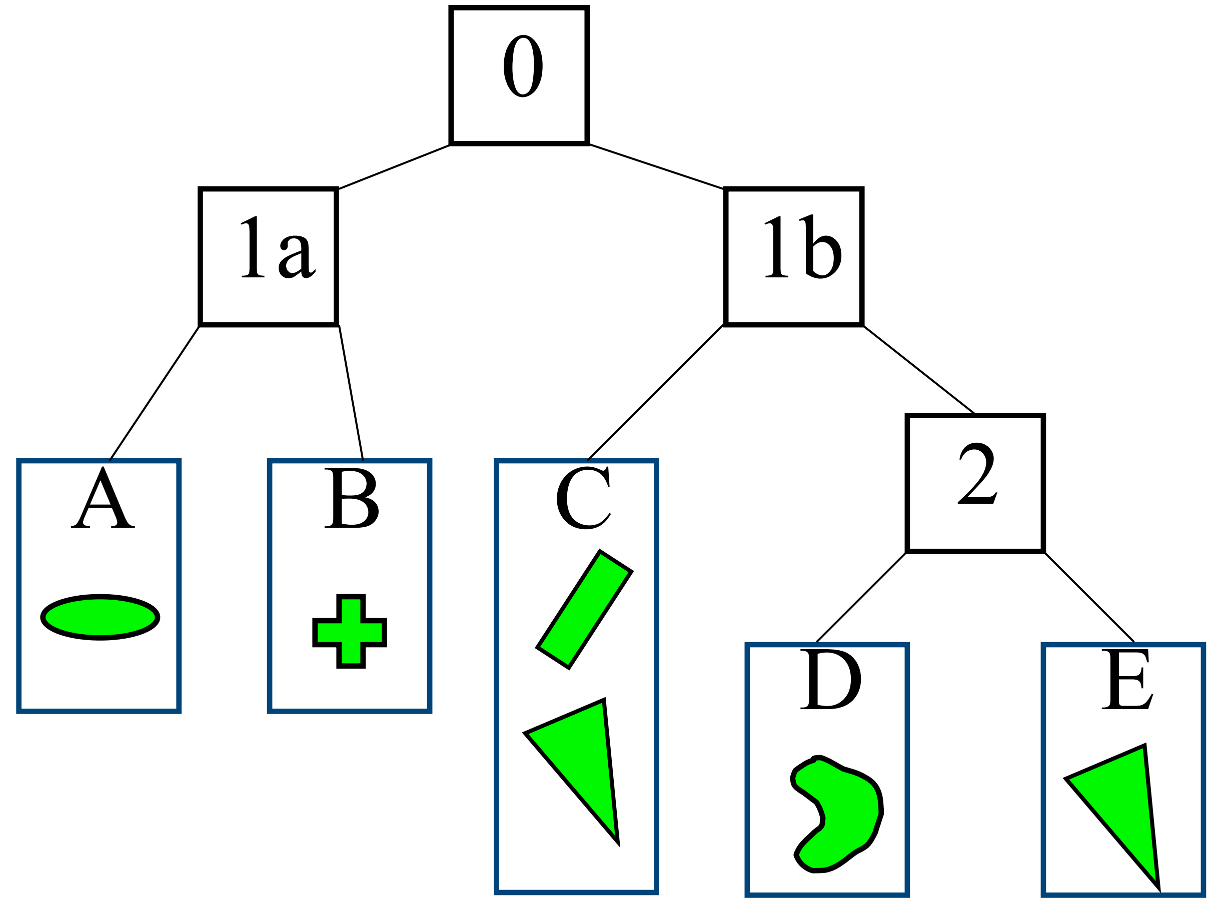
kd-tree(1)

- Can only make a splitting plane along x,y, or z



kd-tree(2)

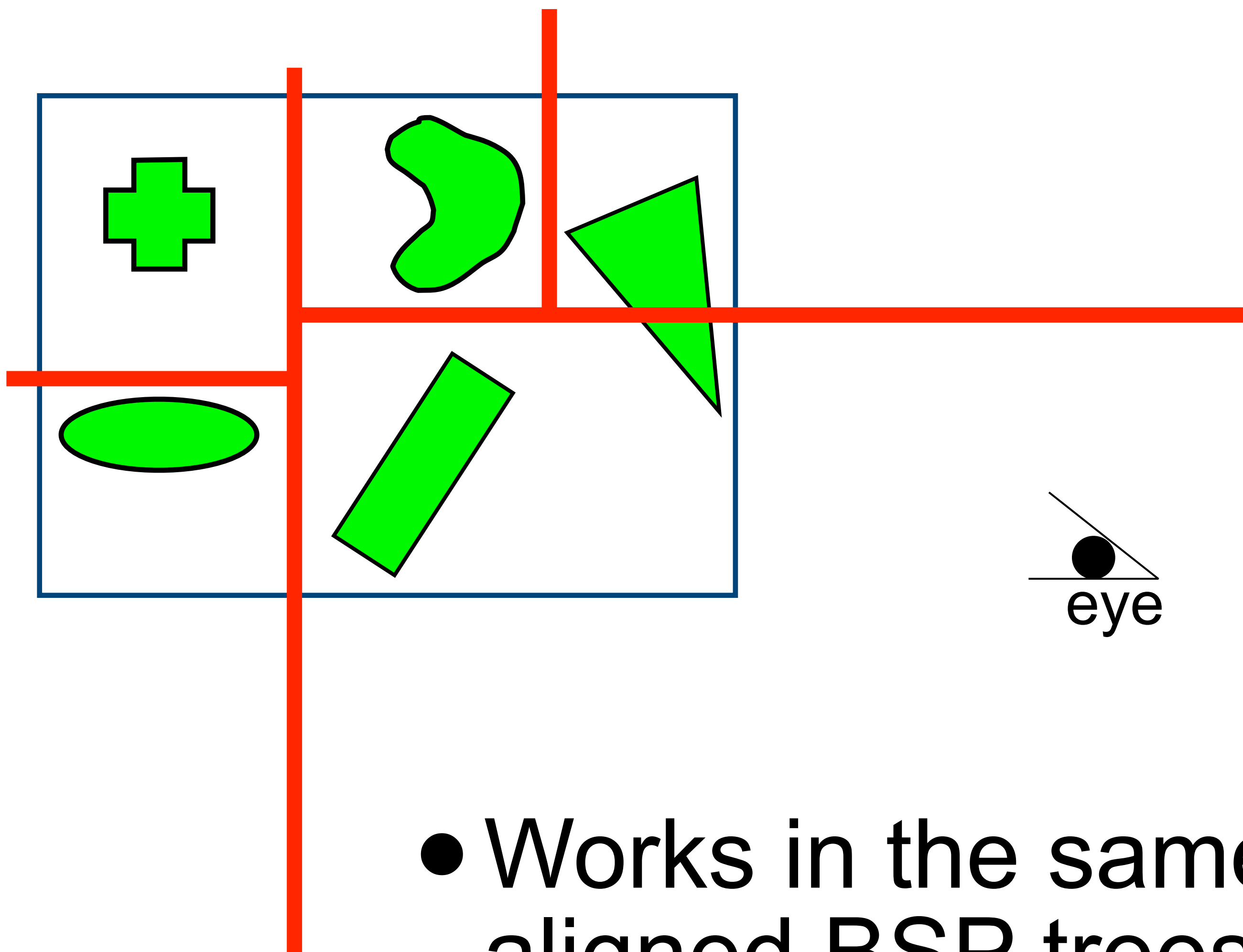


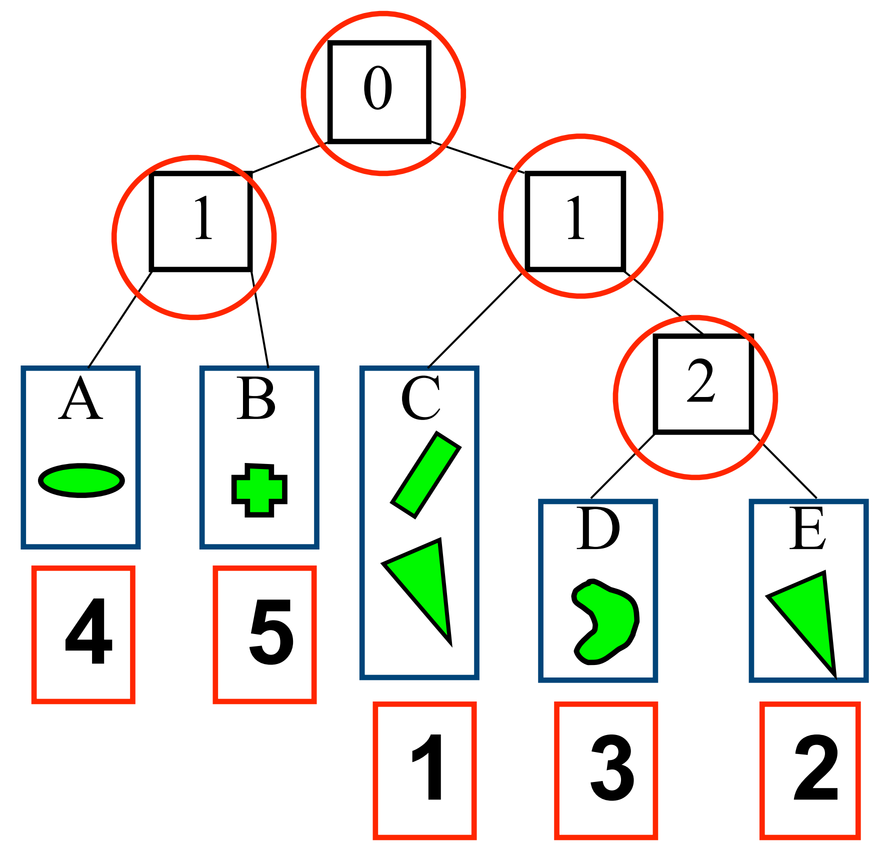


- Each internal node holds a divider plane
- Leaves hold geometry
- Differences compared to BVH
  - Encloses entire space and provides sorting
  - The BV hierarchy can be constructed in any way (no sort)
  - BVHs can use any desirable type of BV

kd-tree
Rough sorting

- Test the planes against the point of view
- Test recursively from root
- Continue on the ”hither” side to sort front to back

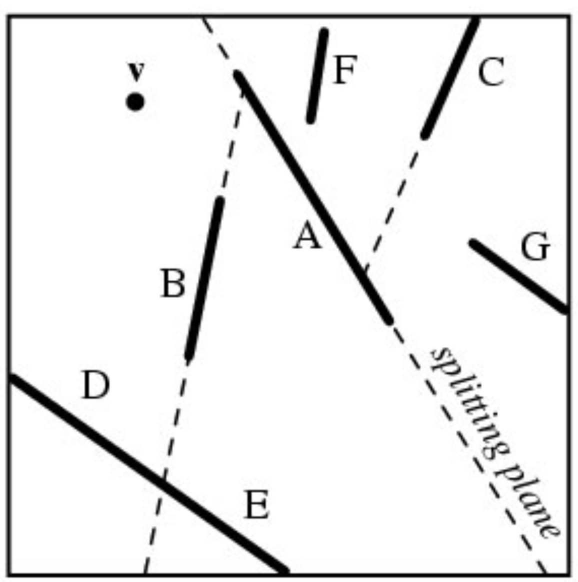


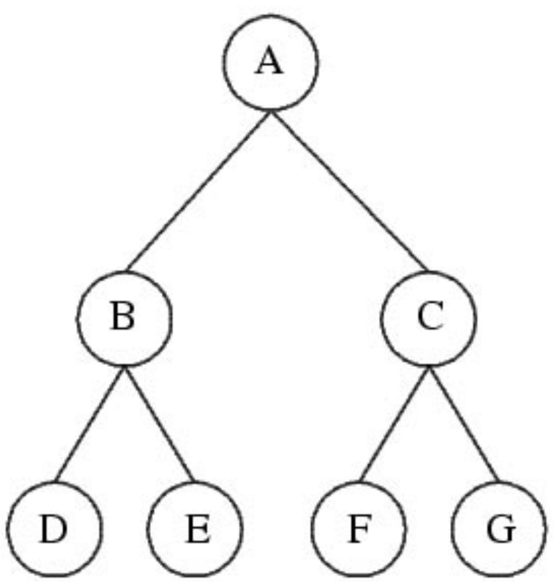


- Works in the same way for polygon-aligned BSP trees --- but gives exact sorting

Polygon-aligned BSP tree

- Allows exact sorting
- Very similar to kd-tree
  - But the splitting plane are now located in the planes of the triangles



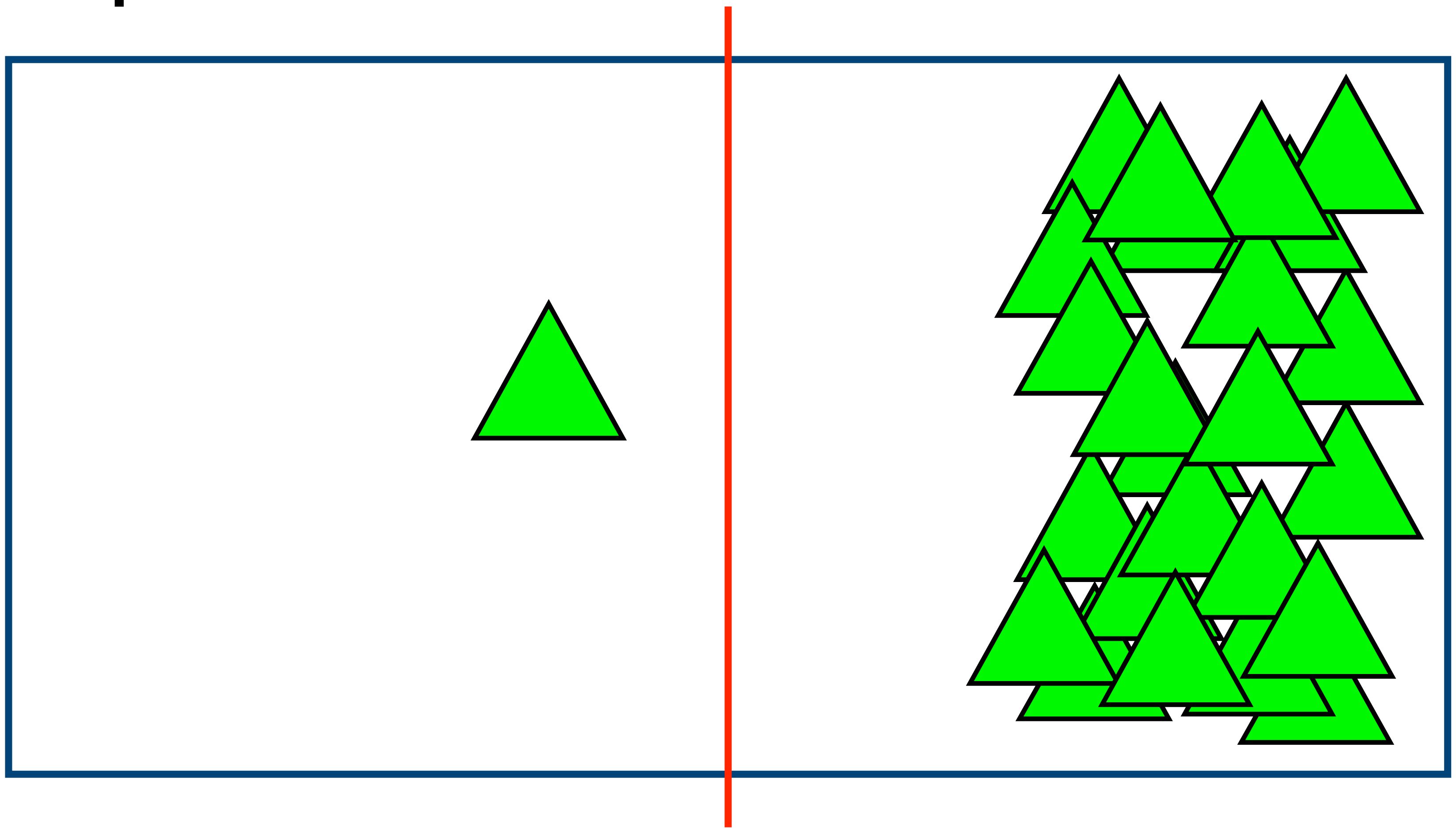


Where to split boxes?

- Mid-point - easy (N.B. Text book uses this)
- Median-split
  - Sort objects along splitting axis by centroid
  - Insert equal number of objects on each side
- Analysis of cost of hitting an object

Where to split?

- Mid point = Bad

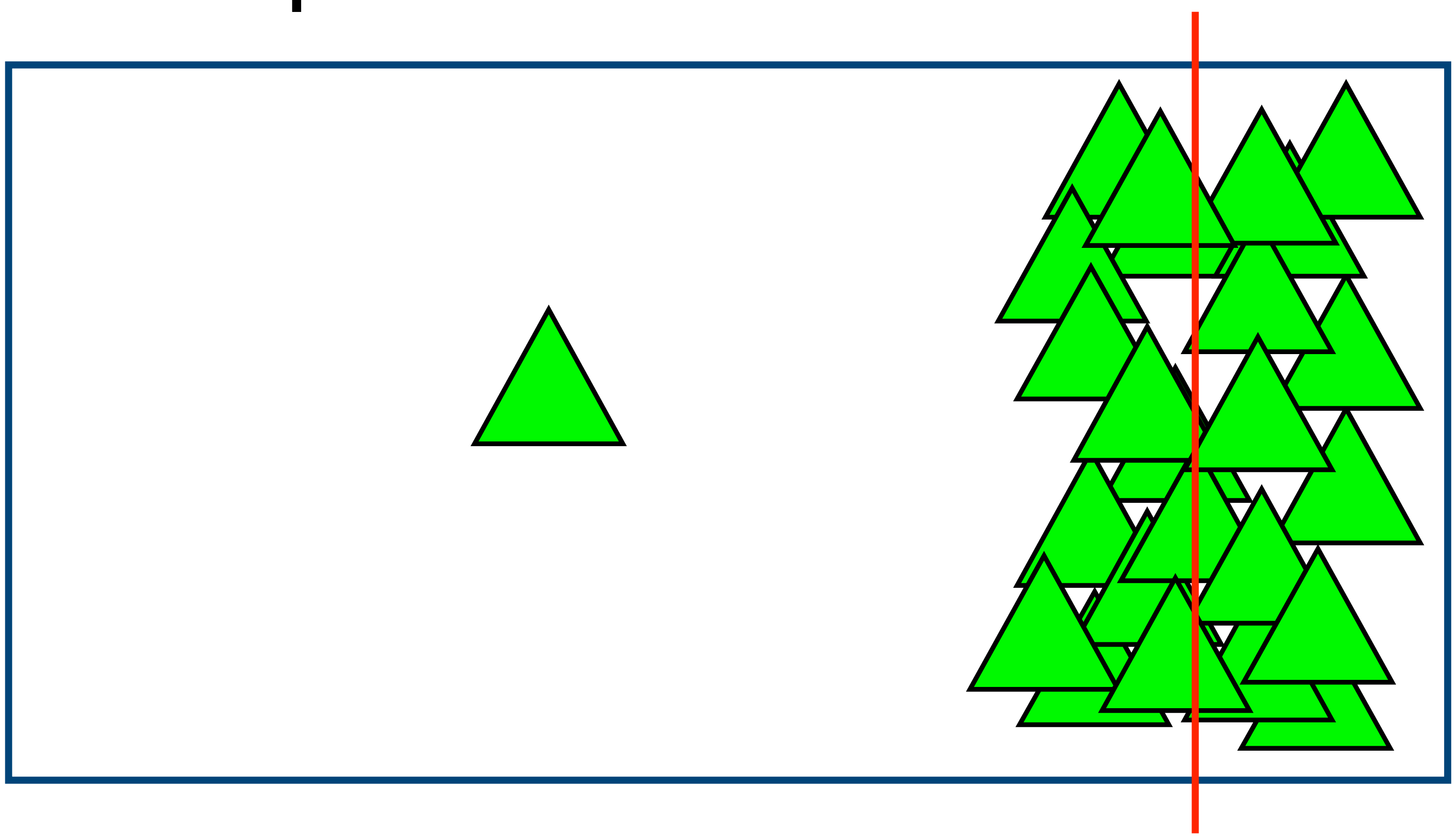


- Makes the L & R probabilities equal
- Pays no attention to the L & R costs

From Gordon Stoll

Where to split?

- Median split = Bad

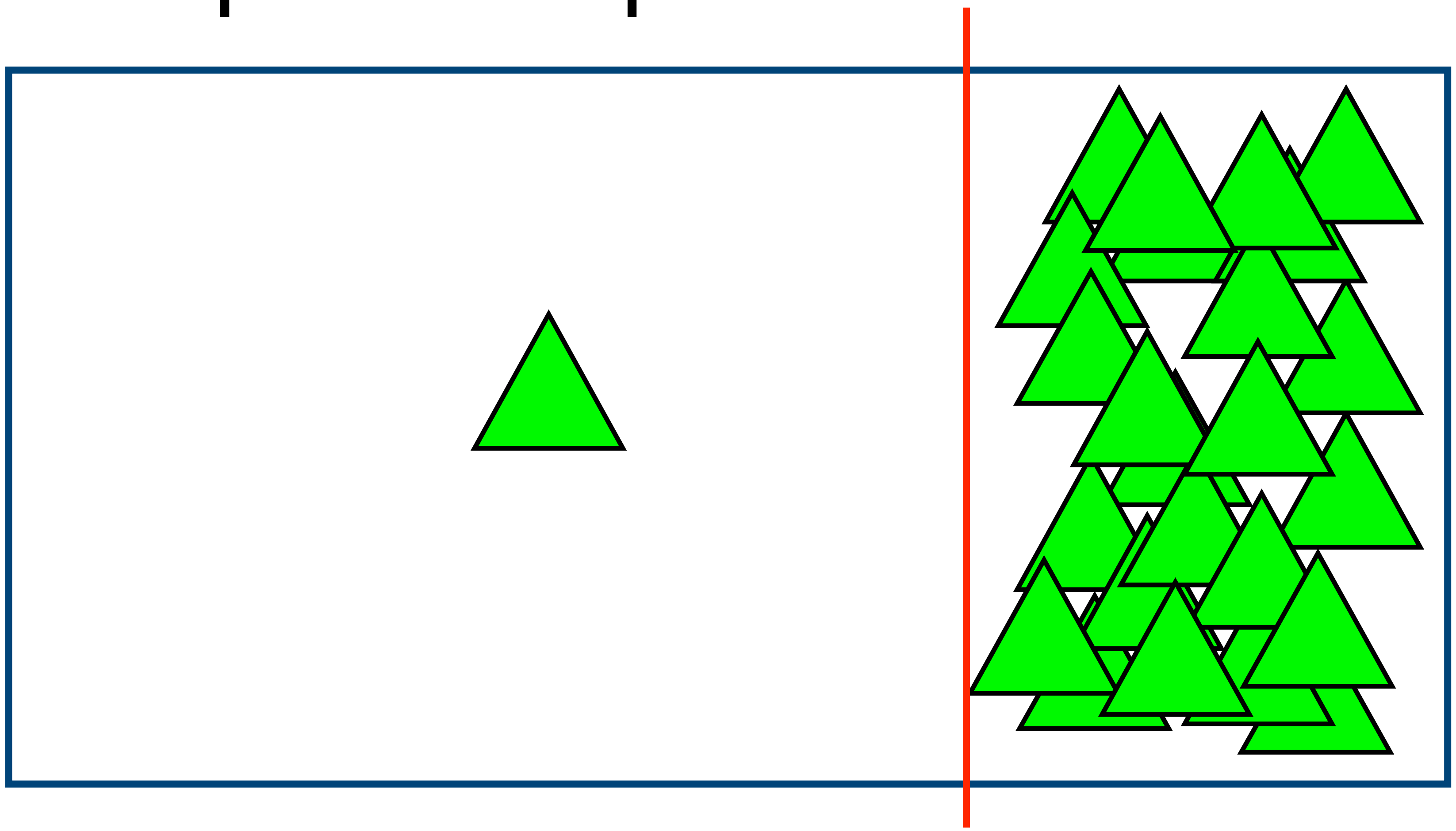


- Makes the L & R costs equal
- Pays no attention to the L & R probabilities

From Gordon Stoll

Where to split?

- Cost-optimized split = Good!

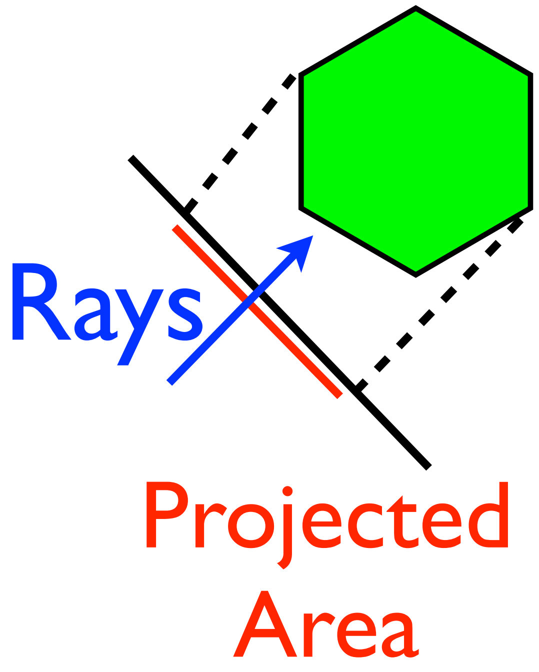


- Automatically and rapidly isolates complexity
- Produces large chunks of empty space

From Gordon Stoll

Cost of nodes

- Cost to trace ray through the node is close to number of triangles
- Number of rays that hit an object from a certain area is proportional to the projected surface area
    - All rays is surface area
- Called Surface-area heuristics (SAH)

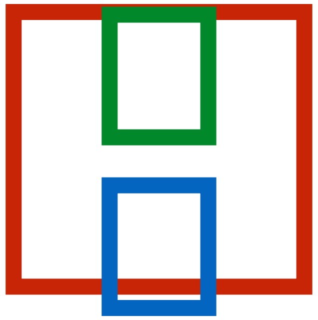


Surface Area Heuristic (SAH)

$$C = C_t + \frac{S_A(B_1)}{S_A(B)}|P_1|C_i + \frac{S_A(B_2)}{S_A(B)}|P_2|C_i$$

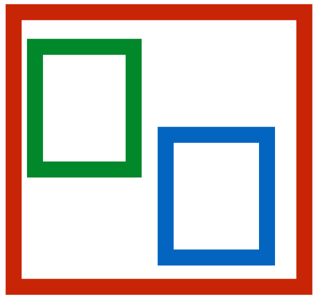
- Cost (C) of tracing a ray through box (B)
  - B_1 and B_2 are child boxes
  - P1 and P2 are number of primitives
  - Ct traversal cost
  - C_i intersection cost
- Compare cost of different split positions
- Terminate when intersecting all primitives is cheaper

Binary Tree Traversal




```
t_min = INF;
stack.push(root);
while (!stack.empty()) {
    currentNode = stack.pop();
    if ( intersect(currentNode) < t_min ) {
        if (currentNode.leaf)
            tmin = intersect(currentNode.objects);
        else
            stack.push(currentNode.children);
    }
}
return t_min;
```

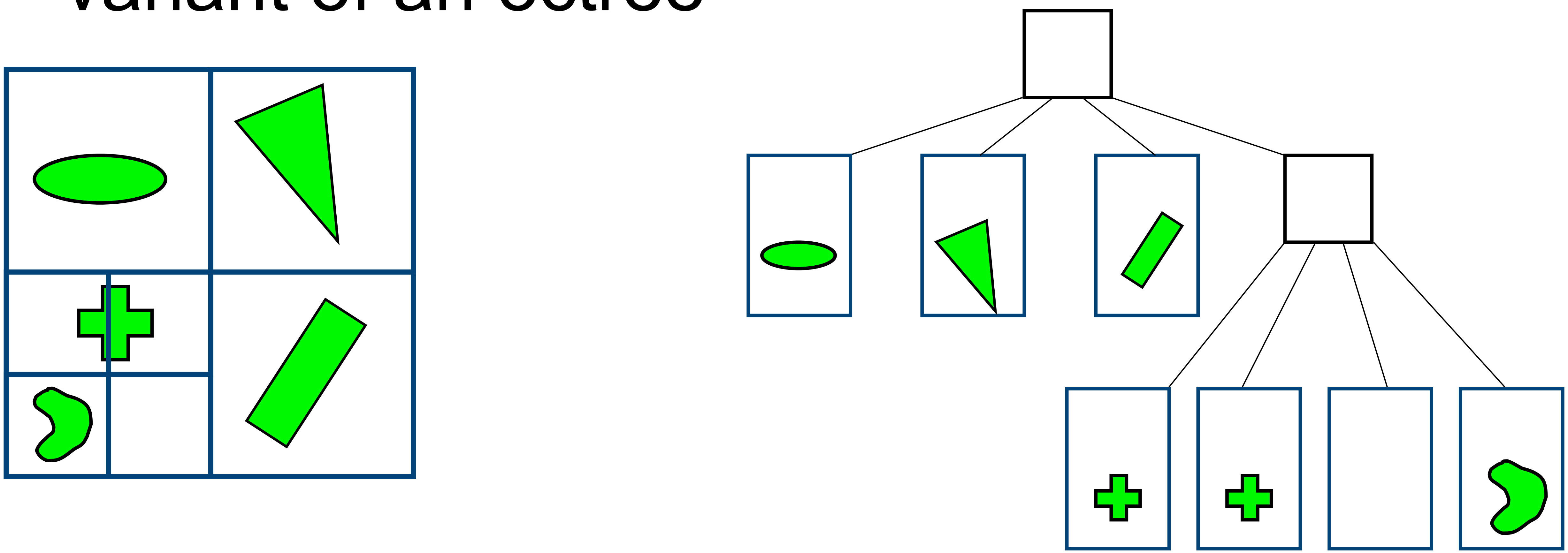
Tree



Optimize!
Put the furthest child on first

Octrees (1)

- A bit similar to axis-aligned BSP trees
- Will explain the quadtree, which is the 2D variant of an octree



- In 3D each square (or rectangle) becomes a box, and 8 children

Octrees (2)

- Expensive to rebuild (BSPs are too)
- Have a uniform structure

- Octrees can be used to
  - Speed up ray tracing
  - Faster picking
  - Culling techniques

Octrees

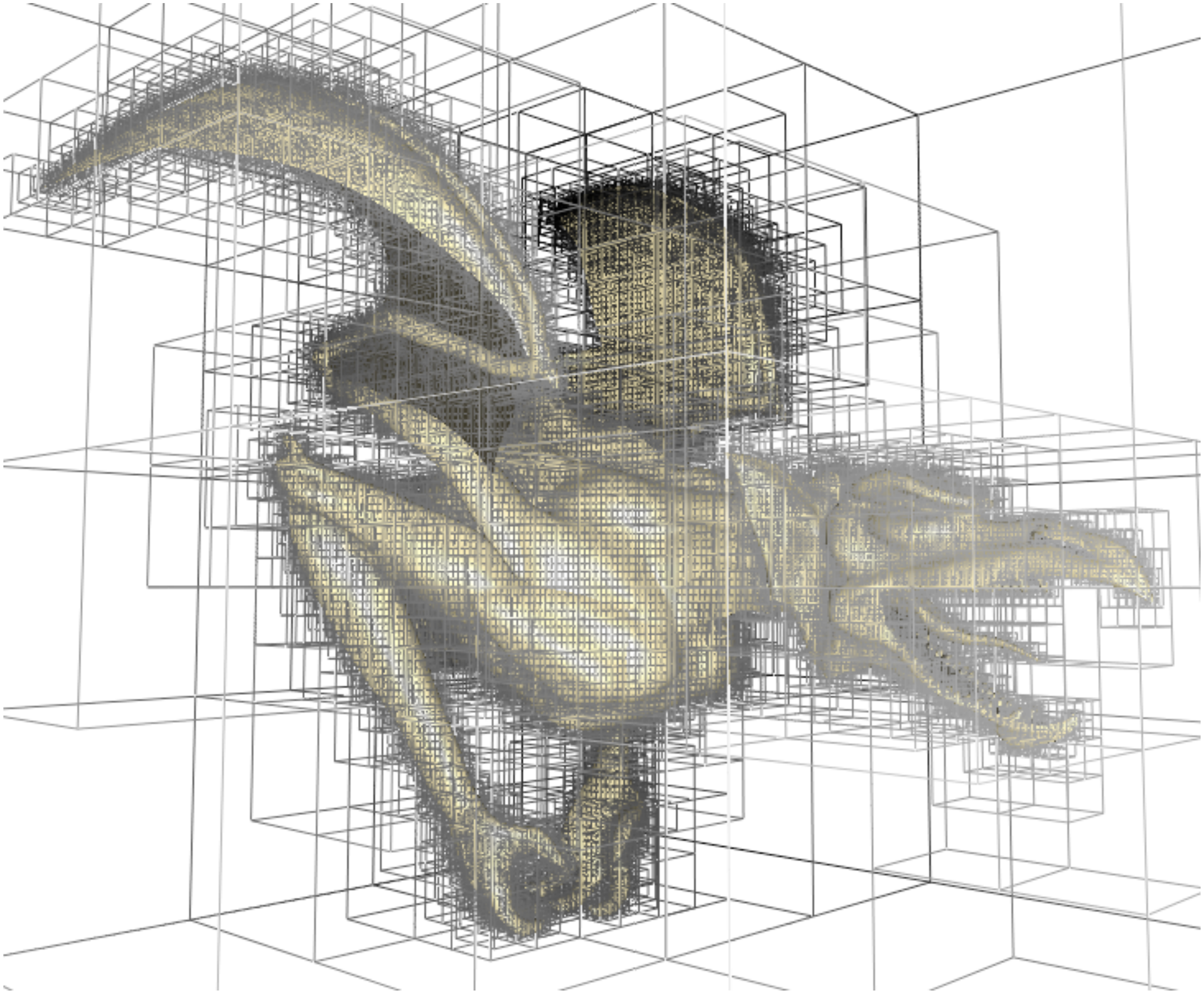
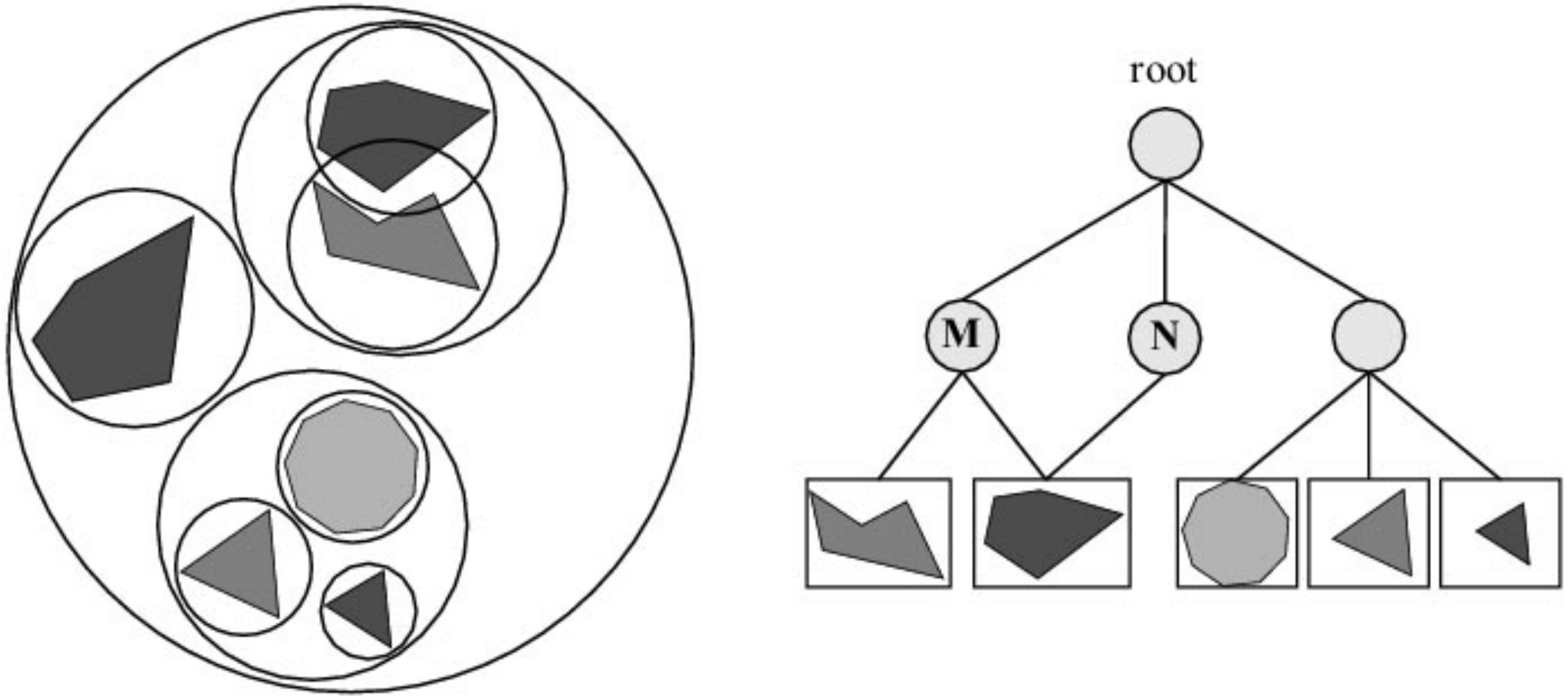


Image courtesy Sylvain Lefebvre et. al., from "Octree Textures on the GPU", GPU Gems2, 2005

Scene graphs

- BVH is the data structure that is used most often
  - Simple to understand
  - Simple code
- However, it stores just geometry
  - Rendering is more than geometry
- The scene graph is an extended BVH with:
  - Lights
  - Textures
  - Transforms
  - And more



Speed-Up Techniques

- Spatial data structures are used to speed up rendering and different queries

- Why more speed?
- Graphics hardware 2x faster in 6 months!
- Wait… then it will be fast enough!
- NOT!
- We will never be satisfied
  - Screen resolution: 4K - 3840 × 2160 8K - 7680 × 4320
  - Realism: global illumination
  - Geometrical complexity: no upper limit!
  - VR - 90Hz, low latency

Data structure summary

- Find intersections faster
- Use simpler objects in hierarchy (AABB)
- Tree type
  - Bounding Volume Hierarchy (BVH)
- Grid based
  - Uniform Grid
  - Octree
- Construction
- Traversal

Next

- Friday Lab, 10-12 or 13-15, sign up on web page
- Questions? Check the forum
- Text book, chapter 9 BVH
- Next week
  - Monday Seminar
    - Lab 2 BVH - Magnus
  - Path tracing!!